I’m a second year Ph.D. at UC Berkeley. My advisor is Prof. Trevor Darrell. I’m majorly interested in computer vision. I’d like to master some parallel programming skills and probably build some fast software to use in computer vision or machine learning.

A potential application of parallel computing is to train deep learning systems, such as convolutional neural networks (CNN) and recurrent neural networks. In the domain of image recognition, such system usually involves a training dataset of over 1 million images[1].

The AlexNet[3] architecture, a seven-layer CNN, takes five to six days to train on two GTX 580 GPUs. A more recent VGG architecture with 19 layers, takes even longer to train. It takes 2 to 3 weeks to train on a system with four Titan Black GPUs. Caffe[4], a powerful deep learning toolbox has recently included both data-parallelism and model-parallelism functions in a recent release[5].

These parallel deep neural network systems usually adopt similar design as Caffe does, although Caffe is not the first to do this. They usually run on a single machine with multiple GPUs. GPU communicates with each other using Nvidia’s P2P DMA access. Usually those applications are written in C++ with the CUDA[12] library.

The large amount of data and computationally more expensive models require running the training process in parallel. However, the use of parallelism is far from satisfactory. Most parallel systems runs with 4 GPUs, because running with more GPUs has diminishing returns. And meanwhile, the time to train a network is far from satisfactory, because 2 weeks is still too long to wait. Not to mention the need to explore various network structures and tuning the parameters.

On the other hand, parameter server[6] aims to provide a general solution for machine learning applications that requires large model size and high training concurrency. Several other systems were also created with the same or similar goals. They are: Graphlab[7], Petuum[8], Mlbase[9] and REEF[10].

However, directly reuse the code for these systems are not straight forward and does not necessarily yields satisfactory speed up. There are several reasons for this. First, training neural network happens on GPU, thus moving data in and out of GPU requires a lot more time, compared to CPU based system. Second, deep neural network is highly non-convex. In comparison, logistic regression and many of its variants, such as sparse logistic regression that is used as test case in [6], are all convex problems. Non-convex problems are much harder to solve than convex ones. Sometimes it’s even hard to converge without adequate initializations and hyper-parameters. In a asynchronous training of the network, it’s even harder to converge, as verified by some preliminary works, such as [11].


